Git 4 Research

Anna Lohmann

2020-01-28

Contents

Chapter 1

Prerequisites

In this introduction we will be mainly using git from the command line (Git BASH).

Whereever beneficial we will using the built-in GUI tools gitk and git-gui.

git-hub will be used for remote repositories.

Should you prefer point and click options for using git you need to find yourself another tutorial.

1.1 NOTE

I will be using pointy brackets '<>' to signify that you have to input whatever your files or names are such as: '', '' or ''. The brackets themselves have to be omitted left when replacing this placeholder.

Chapter 2

Getting started

2.1 Installing git

You will need to install git. How to do this is explained here:

https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

(Please do not opt for git desktop)

This is your go to option for windows: https://gitforwindows.org/

2.2 Initializing git

After the installation there are a few customizations that only have to be done once. They involve telling git your name, your e-mailaddress, your favourite text editor and linking to your git-hub account.

2.2.1 Setting the configuration information

Rightclick anywhere in a filebrowser and click “Git BASH HERE”

This will open a command line where you will have to enter the following information:

‘git config –global user.name ’ ‘git config –global user.email ’

2.3 Getting a git-hub account

Get a GitHub account here: https://github.com/join

If you are a student or a university researcher you might be elligible for a free pro account. Check out your options here:

Chapter 3

Introduction

You can label chapter and section titles using `{#label}` after them, e.g., we can reference Chapter 3. If you do not manually label them, there will be automatic labels anyway, e.g., Chapter **??**.

Figures and tables with captions will be placed in `figure` and `table` environments, respectively.

```
par(mar = c(4, 4, .1, .1))
plot(pressure, type = 'b', pch = 19)
```

Reference a figure by its code chunk label with the `fig:` prefix, e.g., see Figure 3.1. Similarly, you can reference tables generated from `knitr::kable()`, e.g., see Table 3.1.

```
knitr::kable(
  head(iris, 20), caption = 'Here is a nice table!',
  booktabs = TRUE
)
```

You can write citations, too. For example, we are using the **bookdown** package (**?**) in this sample book, which was built on top of R Markdown and **knitr** (Xie, 2015).

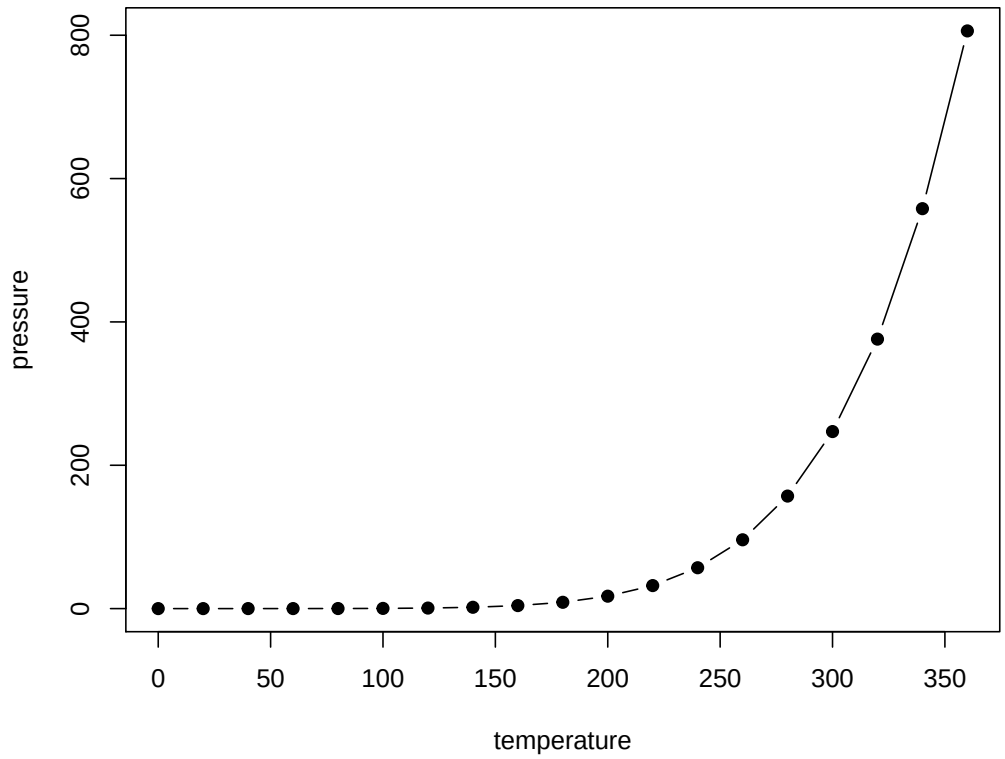

Figure 3.1: Here is a nice figure!

Table 3.1: Here is a nice table!

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5.0	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa
4.9	3.1	1.5	0.1	setosa
5.4	3.7	1.5	0.2	setosa
4.8	3.4	1.6	0.2	setosa
4.8	3.0	1.4	0.1	setosa
4.3	3.0	1.1	0.1	setosa
5.8	4.0	1.2	0.2	setosa
5.7	4.4	1.5	0.4	setosa
5.4	3.9	1.3	0.4	setosa
5.1	3.5	1.4	0.3	setosa
5.7	3.8	1.7	0.3	setosa
5.1	3.8	1.5	0.3	setosa

Chapter 4

The command line

The command line (aka Git BASH or console) is how we tell git what to do. If you are familiar with using a command line in LINUX this one will be just the same. You can change directories, list files and so on.

In this into I will not assume that you have any knowlege of how to use the commandline (i.e. change directories and such) so we will always open it right in the folder we want to use it.

4.1 Open Git BASH

Use your file explorer to navigate to the folder you want to version control with git.

Rightclick anywhere in a filebrowser and click "Git BASH HERE"

This will open a command line where you will have to enter the the commands that you find in this introduciton.

4.2 General notes regarding the commad line

The following should be kept in mind when usind the command line to not go completely crazy in the process: - Capitalization matters - Spacing matters - Some shortcuts like Ctrl+c or Ctrl+v won't work. You can use the right mouse key instead. - The tab key is your best friend as it will auto-complete or partially autocomplete anything for you which greatly reduces the risk for typos - The up-key is your second best friend. Pressing the up-kez will automatically input the last command used. This is especially useful if you got an error and just need to slightly adjust the previous input.

Bibliography

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.